6 Другие средства обработки данных

6.1 Язык awk – обработка текстовых данных CSV

6.1.1 Общая характеристика

awk – специализированный «язык сканирования и обработки шаблонов» (pattern scanning and processing language) и семейство интерпретаторов этого языка.
Другие определения: «язык обработки структурированных текстов», «язык построчного разбора и обработки входного потока» и т.п.

Разработчики: Alfred Aho, Peter Weinberger, Brian Kernigan (1977 г.)

Стандартный компонент Unix-систем, присутствует в большинстве конфигураций.

Последующие версии:

– ***mawk*** – Modified AWK или Mike's AWK – автор Mike Brennan, соответствует POSIX 1003.2 и считается наиболее эффективной

– ***nawk*** – New AWK, версия от AT&T, также используется в FreeBSD

– ***gawk*** – Gnu AWK, является стандартным для Linux, по эффективности реализации сравним с mawk

– ***jawk*** – реализация на языке Java, поддерживает компиляцию скрипта awk в байт-код и ряд специфических для Java-систем возможностей

– ***pawk***, ***pgawk*** – «profiled awk» версии, при меньшей общей производительности формируют выходной файл с «профилирующей» информацией о выполнении программы

и т.д.

awk ориентирован на обработку текстовых данных, в т.ч. в форме CSV, т.е. фактически ***таблиц***.

Характерная особенность – ассоциативные правила применения операций к данным, но сами операции описаны традиционным образом, используя C-образный синтаксис.

6.1.2 Структура входного потока данных

awk предполагает, что входной файл или поток организован как последовательность ***записей*** (***строк***), каждая из которых представляет собой последовательность ***полей***. Разбиение потока на записи и поля – автоматически на основании ***разделителей*** записей и полей (см. ниже). Таким образом, входные данные в интерпретации awk могут рассматриваться как таблица с переменным от строки к строке количеством столбцов, а основными единицами структурирования данных будут ***поток***, ***файл***, ***запись*** и ***поле***.

Поток – все данные, проходящие через awk. Поток может быть безымянным, если он попадает на вход программы путем перенаправления, или состоять из одного или нескольких файлов, если они задаются явным образом. В любом случае поток является непрерывным (однако переход от файла к файлу может быть определен, и их имена будут известны), а интерпретатор автоматически структурирует его по правилам, заданными скриптом или по умолчанию.

Запись (***record***) – строка входного потока или файла, отделенная от предыдущей и последующей символами или комбинациями символов – разделителями записей.

Поле (***field***)– часть записи (подстрока), отделенная от других символами или комбинациями символов – разделителями полей.

Записи и поля – адресуемые.

Адресация записей:
– по ***индексу*** – порядковым номером (начиная с **1**) в потоке и в каждом из файлов потока;
– ***ассоциативно*** – шаблоном, с которым сравнивается строка по правилам регулярных выражений (подобно **`grep`** или **`sed`**).

Адресация полей – по ***индексу*** (начиная с **1**) в текущей записи.

Порядковый номер известен для каждой записи в момент ее обработки и может участвовать в условиях, но произвольное позиционирование на определенную запись невозможно: записи поступают на обработку строго последовательно, и программа не может повлиять на их порядок.

Обращения к полям по индексам обеспечивает прямой доступ к ним, но только в пределах текущей записи.

Структура выходных данных полностью определяется скриптом **awk**. Если фильтры **sed** по умолчанию передают на выход все проходящие через них строки (без изменений или модифицированные), то в **awk** печать выходных данных только явная – соответствующими операторами.

6.1.3 Структура программы awk

Структура программы (скрипта) awk – последовательность ***блоков*** или «***правил***» (***rules***), каждый из которых объединяет ***шаблон*** (***pattern***) и «***действие***» (***action***).

Блок (точнее, его «действие») исполняется, когда выполняется входящее в него условие – шаблон. Так как условие типично проверяет строку входного потока, то, как правило, исполнение всего скрипта представляет собой последовательное применение к каждой строке входного потока тех «действий», условиям шаблонов которых эта строка соответствует. Пустой шаблон соответствует всем записям входного потока.

Блок продолжается до конца строки или включает в себя все строки, заключенные фигурные (операторные) скобки «**{ … }**». Переход на новую строку до скобки «**{**» – начало новой конструкции!

Неправильное оформление блока «**BEGIN**» (предопределенный шаблон, см. ниже) – блок «**BEGIN**» пустой, блок инициализации счетчика повторяется на каждой новой записи потока:

```
BEGIN
  { counter = 1;
  }
```

Правильное оформление блока «**BEGIN**» – инициализация счетчика однократно перед началом обработки потока.

```
BEGIN {
  counter = 1;
}
```

Предопределенные шаблоны:

Шаблон	Выполнение блока
`BEGIN`	до начала обработки всего потока (или всех файлов)
`END`	после окончания обработки всего потока (или всех файлов)
`BEGINFILE`	до начала обработки каждого входного файла
`ENDFILE`	после окончания обработки каждого файла

В свою очередь, блок содержит один или несколько ***операторных строк*** (***statement***), возможно вложенных друг в друга, которые оперируют ***выражениями*** (***expression***)

6.1.4 Типы данных, константы и переменные

Два основных типа данных – ***строка*** и ***число*** (в общем случае вещественное).

Запись числовых констант – по обычным правилам записи чисел, без специальных символов.

Запись строковых констант – в двойных кавычках: «**"*строка*"**». Одинарные кавычки метасимволом не являются.

И переменные, и типизация – ***динамические***:
– тип определяется текущим контекстом
– специального объявления переменных не предусмотрено
– ранее не объявленные переменные создаются при первой попытке ими воспользоваться, при этом автоматически присваиваются значения по умолчанию: пустая строка («») для строк и ноль (**`0`** или **`0.0`**) для чисел.

Обращение к содержимому переменной – без специальных символов (в отличие от shell).

Поддерживаются ***структурированные*** данные – ***массивы***.
В документации обычно именуются «arrays», но в действительности являются хэш-массивами: в роли «индексов» выступают строки (частный случай – строки, представляющие собой числа).

Корректные обращения к элементам массива awk:

`A[0], A[1]` – *индекс – число*
`A["abc"]` – *индекс – строка*
`A[`*`i`*`]` – *индекс – содержимое переменной*

Предопределенные переменные (неполный список):

Имя	Значение по умолчанию	Использование
RS	«New Line» ("**\n**", "**$0A_{16}$**")	символ или строка – разделитель записей (Record Separator) входного потока
FS	«пробел» (“ “, “**20_{16}**”)	символ или строка – разделитель полей внутри записи (Field Separator) входного потока; если не определен, то отдельным полем считается каждый символ
SUBSEP	ASCII **28** (**034_8**)	разделитель элементов составного «индекса» (multiple subscripts) многомерного массива и вообще под-полей

ORS	равен RS	разделитель записей в выходном потоке (Output RS), подставляется автоматически
OFS	равен FS	разделитель полей в выходном потоке (Output FS), подставляется автоматически
OFMT	**%.6g**	формат вывода чисел для оператора **print** (см. ниже)
CONVFMT	**%.6g**	форматная строка для автоматического преобразования между строками и числами
NR	–	номер текущей записи во входном потоке, счетчик обработанных записей в потоке (Number of Records)

FNR	–	номер текущей записи в текущем файле, счетчик обработанных записей в файле (File Number of Records)
NF	–	количество полей в текущей записи (Number of Fields)
FILENAME	–	имя текущего обрабатываемого файла; если файл не определен, то имя равно **"-"**; в блоке **BEGIN** имя не определено, если только не использовался оператор **getline**.
ARGC, **ARGV**	–	количество аргументов командной строки и массив их значений, индексируемый с **0** (аналогично C-программам)

ENVIRON	–	массив текущих переменных окружения, «индексами» которого являются их имена
$0	–	текущая запись входного потока без разделения на поля
$1, **$2**, …	–	отдельные поля текущей записи по их индексам (начиная с **1** и без ограничения максимального). Индекс может быть переменной, например ***$i*** – ***i***-е поле текущей записи

Использование **ARGC** и **ARGV** затруднено. Первым по счету аргументом (**ARGV[0]**) является имя самого интерпретатора, а последующими (**ARGV[1]**, **ARGV[2]** и т.д.) – те аргументы командной строки, которые не были им распознаны опции. В частности, аргументами становятся имена файлов, передаваемых интерпретатору. При этом интерпретатор все равно пытается рассматривать все эти аргументы как файлы, поэтому использование иных, отличных от имен доступных файлов значений, приведет к ошибкам выполнения. При попытке просмотреть содержимое **ARGV** циклом-итератором будут получены не значения аргументов, а их индексы (следствие реализации массивов в awk).

6.1.5 Арифметические и логические операции

Поддерживаемый набор операций и их синтаксис практически совпадает с языком C, включая инкремент, декремент (префиксная и постфиксная форма различаются) и совмещение арифметической операции с присваиванием.

Для чисел дополнительно могут быть определены операции «^» или «**» – возведение в степень (зависит от версии awk).

Для строк дополнительно определена операция «~» и «!~» – проверка на соответствие регулярному выражению (в дополнение к обычному точному сравнению).

Для объединения (катенации) строк достаточно записать их без разделителя, специальная операция не предусмотрена.

Битовые операции не предусмотрены.

6.1.6 Управляющие конструкции

Конструкции ветвления **`if - then`** и циклов **`while`** практически не отличаются от аналогичных в языке C.

```
if (NR == 1) { … } #действия для 1-й строки потока
else { … } #действия для остальных строк потока
```

Цикл **`for`** существует в двух разновидностях: со счетчиком (аналогично языку C) и с итератором.

Пример счетчиковой версии for:

```
for (i=1; i<=NF; ++i) {
  FldNames[i] = $i; #массив имен полей таблицы
}
```

Пример итерационной версии for:

```
for (i in FldNames) {
  print i FldNames[i];
} #печать номеров и имен полей из массива
```

Здесь переменная ***i*** получает поочередно значения «индексов» хэш-массива, а не соответствующих им значений!

6.1.7 Операторы ввода-вывода

В общем, подобны библиотечным функциям, но имеют отличия в синтаксисе вызова – не требуют скобок, а выглядят как команда со списком параметров:

`print` ***`x1 x2 x3`*** **…** – *простое перечисление выводимых значений*

`printf` ***`format x1 x2 x3`*** **…** – *форматный вывод*

Форматная строка **`format`** в общем аналогична используемой в функциях семейств **`printf()`** в стандартной библиотеке C.

Оператор ввода также подобен функции:

`getline`

6.1.8 Функции

Место функций – обычное для языков программирования. Синтаксис – в целом аналогичен языку С, но надо учитывать особенности динамической типизации и динамически создаваемых объектов:

***имя_функции*(*параметры*)**

Обширная библиотека функций различного назначения: работа с числовыми данными, строками, ввод-вывод и т.д.

Например, некоторые библиотечные функции:

Функция	**Эффект**

Строковые:

length(*str*) – *длина строки*
index(*str1*,*str2*) – *поиск подстроки в строке*
match(*str*,*regexp*) – *поиск соответствия регулярному выражению*
sub(*for*,*repl*,*str*) и gsub(*for*,*repl*,*str*) – *замена* ***for*** *на* ***repl*** *в строке* ***str*** *(аналогично команде* **s** *в* **sed**)
sprintf(*str*,*format*,*x1*,*x2*,…) – *«печать» в строку (аналогично* **sprintf()** *в* **C**

Математические:

exp(*x*), log(*x*), cos(*x*), sqrt(*x*)

и т.д.

Помимо готовых библиотечных функций, можно определять новые (пользовательские). Функции описываются в любом месте текста программы, но вне любого блока, и становятся глобальными – доступны во всех последующих блоках.

Описание функции:

function *myfunc*() (*params*) { *implementation*; return *result* }

Например:

При вызове функции допустимо передавать неполный список параметров. Все параметры, которые были объявлены как формальные (присутствуют в описании функции), но не получили фактических значений, инициализируются как пустые и считаются локальными переменными функции. При этом синтаксического различия между локальными переменными и фактически переданными параметрами внутри функции нет.

Переменные, создаваемые внутри функций, также становятся глобальными – доступны вне функции. Аналогично, глобальные переменные, объявленные вне функции и известные на момент ее вызова, доступны и внутри нее.

6.1.9 Примеры

Матрица чисел

вычисление суммы всех числовых значений, входящих в матрицу, и их среднего значения. Разделителем полей считается символ табуляции – «`\t`». Строки, начинающиеся с символа «#», считаются комментариями, они выводятся на консоль и в вычислениях не участвуют (символы «пробел» и «табуляция» в начале строки игнорируются). При обнаружении нечислового поля в поток staderr выводится сообщение об ошибке.

```
BEGIN {
 FS = "[ \t]"
 Cnt = 0; Sum = 0.0
}
/^[ \t]*#/ { #commented rows
 print "Comment line - " $0
}
!/^[ \t]*#/ { #all rows excluding commented-out
ones
 for (i=1; i<=NF; ++i) {
    if ($i ~ "^#") break #commented-out part of the
row
    if ($i ~ "^[-+0-9.]+$") { Sum += $i; ++ Cnt }
    else print "Wrong field: " $i > "/dev/stderr"
 }
}
END {
```

```
  Avg = (Cnt > 0) ? (Sum / Cnt) : 0.0
  printf "Cnt = %i\tSum = %f\tAverage = %f\n", Cnt,
Sum, Avg
}
```

Количество столбцов в строках не контролируется. Также упрощено регулярное выражение, классифицирующее значение очередного поля как числовое: оно анализирует только состав символов поля, но не их порядок; в ряде случаев поле будет считаться «числовым», но преобразование его в число закончится неудачей, и будет подставлено нулевое значение. Усовершенствование проверок входных данных предлагается в качестве самостоятельного упражнения.

Работа с байтовым потоком

Разбор записей и полей фиксированной ширины, но без разделителей (т.е. не CSV-формат). Работа с массивами.

```
BEGIN {
 OFS = "\t"
 id_offs = 256; id_width = 4
 rec_width = 2048
}
{
 l=length( $0)
 for (i=id_offs; i<l; i+=rec_width) {
    id = substr( $0, i, id_width) #извлечение
значений
    sub( "^ +", "", id); sub( " +$", "", id)
#нормализация
    if (id != "") ++ Ids[id] #= Ids[id] + 1
#сохранение уникальных со счетчиком
 }
}
END {
```

```
  for (id in Ids) print id, Ids[id] #вывод таблицы
результатов
}
```

Здесь предполагается, что входной файл (поток) состоит из единственной «строки», не имеющей разделителей и содержащей одну или несколько «записей» известной «ширины» (в примере 2048 байт). Разделять записи приходится искусственно, так как для awk файл без разделителей воспринимается как единая запись. Интересующее «поле» лежит по заданному смещению в каждой «записи» (в примере 256) и имеет заданную ширину (в примере 4 байта).

Для упрощения «записи» как таковые из потока не выделяются, переменная создается только для интересующего «поля». Найденные значения нормализуются путем отбрасывания пробелов, пустые значения игнорируются. С помощью массива со строковой индексацией организуется сохранение только уникальных значений и подсчитывается число их обнаружений в потоке данных. Результаты выводятся без сортировки.

Генератор одиночного случайного числа.

Библиотека awk предоставляет готовый генератор случайных чисел и пару функций: **`srand()`** инициализирует генератор, **`rand()`** формирует очередное значение. Для правильной работы генератора необходимо, чтобы значение «затравки» само было случайным; для этого удобно использовать счетчик секунд, имеющийся в системе и доступный с помощью команды **`date`** с соответствующими параметрами.

Библиотечная функция **`system()`** позволяет выполнить внешнюю команду, но остается проблема получения ее результата – строки, выведенной в **`stdout`**. Самым простым решением будет воспользоваться средствами shell, объединив в одной командной строке **`date`**, **`awk`** и «скрипт» для него:

```
awk 'BEGIN { srand( ``date +%s`') ; print rand() }'
< /dev/null
```

Здесь shell выполняет команду **`date`**, подставляет выведенное в stdout значение внутрь командной строки и далее вызывает **`awk`**, передавая ему эту строку. Командная строка («скрипт») **`awk`** оказывается разбитой на части: фиксированные фрагменты, экранированные одинарными кавычками, и вставленный между ними вызов внешней команды в обратных кавычках; так как пробелов между фрагментами нет, они сливаются в единую строку. Команда **`date`** с параметром «**`+%s`**» выводит единственное числовое значение – счетчик секунд от «начала эпохи». «Скрипт» **`awk`** состоит из единственного блока **`BEGIN`**, т.е. обработка входных строк не предполагается. Тем не менее, для немедленного останова после выполнения этого блока выполняется переключение ввода на «пустое устройство» – тем самым поток ввода будет закрыт. Сгенерированное значение выводится в **`stdout`** в естественном вещественном формате.

Экранирование «скрипта» в командной строке, промежуточное хранение счетчика в переменной shell и другие технические подробности реализации могут быть решены по-разному.

Более логичная и «прозрачная» реализация:

```
date +%s | awk '{ srand( $1); print rand() }'
```

Здесь **date** вызывается отдельно, и ее результат перенаправляется во входной поток **awk**. «Скрипт» в этом случае включает единственный блок, применяемый к единственной входной строке. Опционально можно добавить фильтрацию строк регулярным выражением, однако можно рассчитывать, что ее формирование полностью контролируется командой shell и всегда будет корректным.

6.2 Программируемый калькулятор bc

Интерпретатор специализированного C-образного языка для математических вычислений.

Вход – программа (скрипт) в виде файла, в **`stdin`** или непосредственно в командной строке; данные в **`stdin`**.

Выход – данные в **`stdout`**, явный или неявный вывод

Некоторые опции командной строки:

Опция	Эффект
`-l`	подключение математической библиотеки
`-s`	использовать только стандартный (POSIX без расширений) язык
`-q`	подавление вывода вспомогательной (справочной) информации о **`bc`** («GNU **`bc`** welcome»)

6.2.1 Типы данных и константы

Основной тип – вещественное число, но в строковом представлении в заданной системе счисления и управляемой точностью (***arbitary precision***), т.е. с заданным количеством знаков после запятой, причем изменение точности в ходе выполнения программы на уже вычисленные значения не влияет.

Строковые константы (литералы) – заключенные в двойные кавычки: **"*строка*"**.
Одинарные кавычки – обычный символ в строке, специального смысла не имеет.

6.2.2 Переменные

Скалярные (обычные) и массивы (поддержка частичная).

Неинициализированные переменные считаются нулевыми.

Имена переменных (и функций) – в стандартной версии односимвольные в нижнем регистре. Расширенный синтаксис допускает многосимвольные имена (но требует буквы по-прежнему в нижнем регистре).

Некоторые предопределенные переменные:

Имя	**Значение по умолчанию**	**Использование**
`scale`	0 (без мат. библ.) 20 (с мат. библ.)	точность представления чисел – количество цифр в дробной части
`ibase`	**`10`**	основание системы счисления чисел во входных данных
`obase`	**`10`**	основание системы счисления чисел в выходных данных
`last`	**`–`**	последнее выведенное значение

6.2.3 Арифметические и логические операции

В основном повторяют используемые в языке С. Дополнены операцией возведения в степень: «^».

Приоритеты операций – в основном аналогично С, но присваивание («=») не имеет высшего приоритета.

6.2.4 Управляющие и другие «специальные» конструкции

Практически не отличаются от используемых в языке С:

Ветвление: **`if`** – **`else`**

Циклы: **`while`**, **`for`** – обычные. Также **`break`** и **`continue`**.

Остановка выполнения программы: **`halt`**. Работает только в реально выполняемой ветви программы.

Остановка программы даже в не выполняемой реально ветви, т.е. немедленно при обнаружении интерпретатором: **`quit`**.

6.2.5 Вывод значений

Вывод списка значений: **`print`** ***`список_значений`***

Вывод строки: **`string "`*`строка`*`"`**

Результат вычисления выражения, если он не присвоен переменной и не использован в составном выражении, по умолчанию выводится в **`stdout`**.

6.2.6 Функции

Есть библиотеки функций – стандартная и подключаемая математическая (опция «**-l**»), поддерживаются также пользовательские функции

Некоторые «стандартные» функции:

Функция	Эффект
length(*x*)	число значащих цифр в значении переменной, выражения
scale(*x*)	число цифр в дробной части значения
sqrt(*x*)	квадратный корень
read(*x*)	ввод значения переменной из **stdin**. Создает конфликт с вводом через **stdin** строк программы

Некоторые функции математической библиотеки:

Оператор	Эффект
s(*x*)	синус (угол в радианах)
c(*x*)	косинус (угол в радианах)
a(*x*)	арктангенс (результат в радианах)
e(*x*)	показательная функция – e^x
l(*x*)	логарифм (натуральный)

Функция, определяемая пользователем:

```
define имя_функции ( параметры ) {
  auto список_локальных_переменных
  ... список_операций
  return результат
}
```

6.2.7 Комментарии

В стандартной (POSIX) версии языка – экранированные с обеих сторон, аналогично классическому C:

```
… /* комментарий */
```

В расширенной версии – также и однострочные:

```
… # комментарий
```

6.2.8 Примеры

Пример: простейшая функция удвоения числа

```
define d(x) { return (2*x); }
```

Пример: рекурсивная функция вычисления факториала

```
define f(x) {
  if (x<=1) return 1;
  return (f(x-1)*x;
}
```

Пример: вычисление числа «пи»

Командная строка shell с непосредственной передачей в **bc** его «программы». Точность результата не очень высокая:

```
pi=`echo "scale=10; 4*a(1);" | bc -l `
```

Пример: заполнение таблицы числовых значений

Скрипт **bc** из реального проекта – формирование таблицы значений счетчика-делителя для генератора звукового сигнала.

```
scale = 12

pi = 4.0 * a(1)

define pow( x, y) { return e(y*l(x)) }

define f_to_cnt( f) {
# 2MHz source clock, 8-phase signal shape
  scale_save = scale
  scale = 1
  cnt = 2000000 / 8 / f
  scale = scale_save
  return cnt
}
```

```
define sin_pow_table( \
   f_min, f_max, p_0, n_per, n_int) {
#f_min to f_max value range,
#p_0 to (p_0+n_per)(*Pi) phase,
#n_int intervals (n_int+1 points)
 x = p_0*pi #arg for sin (initial phase to begin)
 h = (n_per*pi)/n_int #increment step for x (phase)
 k = pow((f_max/f_min),(1.0/n_int))
 for (i=0; i<=n_int; ++i) {
   y = ((s(x) + 1.0) * n_int / 2) #arg for pow
   f = f_min * pow(k,y)
   cnt = f_to_cnt(f)
   print i, ":\t", f, "\t", cnt, "\n"
   x += h #arg for sin
 }
 return(0)
}
```

```
#-- 1410 to 2820 Hz (16 intervals), then
#2820..2420 Hz cyclic (7.5 intervals per period)
res = sin_pow_table( \
   (2620+200)/2, (2620+200), -0.5, 1.0, 16 )
print "\n"
res = sin_pow_table( \
   (2620-200), (2620+200), 0.5, 2.0, 7 )
print "\n"
res = sin_pow_table( \
   (2620-200), (2620+200), 0.5, 2.0, 8 )
quit
```

Частота сигнала возрастает от начальной до околорезонансной и затем колеблется вблизи нее. Закон изменения частоты синусоидальный, с геометрическим шагом приращения (множитель вместо фиксированной абсолютной величины). Таблица составляется из трех частей.

6.3 Язык perl (основы)

6.4 Локализация программного обеспечения

6.4.1 Общие сведения и концепции

Правильнее говорить шире – об интернационализации и локализации.

Интернационализация – методика и технология проектирования и реализации ПО, которое бы допускало (поддерживало) бы адаптацию в соответствии с региональными требованиями без существенной переработки (способность работать с разными ***локалями***).

Локализация – процесс адаптации ПО в соответствии с региональными требованиями. Например, для использования локализованного продукта достаточно будет изменить ограниченный набор системных настроек-«переключателей» (приспособление к конкретной ***локали***).

(Часто используются сокращения терминов: «i18n» вместо «internationalization» и «l10n» вместо «localization». Можно встретить в именах файлов, переменных и т.п.)

Уровни интернационализации (в реальности сложнее):

– уровень шрифтов и кодировок (минимум – «чистая» поддержка кодировки Latin-1)

– уровень форматов (ввода-вывода) и сравнения строк

– уровень сообщений и представления текста

– уровень поддержки «азиатских» языков (многобайтная кодировка)

Уровни (этапы) локализации:

– обеспечение поддержки языка (шрифт, кодировка) и стандартов представления данных

– перевод текстов (сообщения, справка и проч.) на целевой язык

– тонкая настройка на региональные требования

Интернационализация и локализация могут предусматривать адаптацию не только текстовых строк (сообщений), но и цветовых схем, визуальных образов и т.п.

6.4.2 Некоторые стандарты и стандартизирующие организации

LISA (Localization Industry Standards Association) – с 1990 по 2011 гг. международная ассоциация клиентов и поставщиков услуг в области глобализации и локализации. После роспуска стандарты переданы в открытый доступ.

GALA (Globalization and Localization Association)

OASIS (Organization for the Advancement of Structured Information Standards)

XLIFF (XML Localization Interchange File Format) — основанный на XML расширяемый платформенно-независимый стандарт для обмена локализуемыми данными и сопутствующей информацией

ISO 639 – набор стандартов ISO, связанных со стандартизацией условных обозначений языков и языковых групп. В частности, определяет языковые коды: ISO 639-1 – двухбуквенные (например, **en**), ISO 639-2 – трехбуквенные (например, **eng**)

ISO 3166 – еще один набор стандартов ISO, связанных со стандартизацией условных обозначений стран и территорий. В частности, определяет коды: ISO 3166-1-alpha2 – двухбуквенные (например, **GB**), ISO 3166-1 alpha-3 – трехбуквенные, ISO 3166-1 alpha-3 – трехзначные числовые коды

RFC (Request For Comments) – множество документов, устанавливающих фактические стандарты, связанные с сетями. В частности, RFC 3066, 4646, 4647 – идентификаторы языков

Языковые ***теги*** и ***коды*** составляются на основе стандартизованных обозначений и обычно включают: идентификатор языка (языковой семьи), идентификатор территории, идентификатор кодировки. Например, строка

`ru_RU.UTF-8`

представляет версию русского языка (либо, в зависимости от контекста, иные региональные настройки), используемую в РФ, в сочетании с Unicode

6.4.3 Обеспечение интернационализации/локализации на уровне системы

Локаль (***locale***) – набор параметров, задающих «региональные» настройки, в первую очередь пользовательского интерфейса.

Можно представить локаль как специализированную базу данных, которая содержит правила и соглашения:

– классификация символов и преобразований;

– представление «денежных» величин («экономический» формат);

– представление прочих («обычных») чисел;

– форматы даты и времени.

Это формализуется как ***категории***. Список категорий не фиксирован и может расширяться в зависимости от системы. Базовый (обязательный) набор категорий:

– **LC_ALL** – покрывает все параметры локализации;
– **LC_COLLATE** – для функций **strcoll()** (сравнение с учетом локали) и **strxfrm()** (преобразование строки в соответствии с настройками локали)
– **LC_CTYPE** – для функций преобразования типа
– **LC_MONETARY** – «экономический» (финансовый) формат представления
– **LC_NUMERIC** – числовой (десятичные цифры)
– **LC_TIME** – формат даты и времени (функции **strftyme()** и **wcstrftime()**)

Категориям присваиваются теги из общего пространства значений (например, «**`ru_RU.UTF-8`**»), и в зависимости от «специализации» данной категории настройка начинает действовать на соответствующую часть функционала. Порядок просмотра источников значений для категорий:
– переменная **`LC_ALL`** – применяется, если она определена и не пуста;
– иначе применяется значение переменной с соответствующим именем, если она не пуста;
– иначе применяется содержимое переменной **`LANG`**.

Традиционно для Unix-систем, конфигурация локали хранится в файле (файлах).

Также традиционно предусмотрен интерфейс к настройкам – команда **`locale`**. В зависимости от опций дает доступ к списку настроек или к отдельным настройкам.

Например, для хост-машины «mars401» (192.168.1.15) под ОС OpenSUSE Linux:

```
> echo $LANG
  ru_RU.UTF-8
```

Т.е. устанавливается использование русского языка и «русских» прочих настроек в версии, принятой в РФ, с кодировкой Unicode UTF-8. При этом установка выполняется через переменную **LANG**, а специализированные категории определены как «пустые».

Файл **/etc/sysconfig/language** – определены переменные **RC_LANG**, **RC_LC_ALL**, **RC_LC_CTYPE**, **RC_LC_NUMERIC** и т.д. Они будут использоваться для инициализации соответствующих переменных-категорий.

Частично данные о локализации включены и в другие файлы. Например, файл **`/etc/sysconfig/language`** – строка **`CONSOLE_ENCODING="UTF-8"`** – кодировка для вывода не-ASCII символов.

В результате получаем набор категорий, доступный, например, посредством команды **`locale`**:

```
> locale
  LANG=ru_RU.UTF-8
  LC_CTYPE="ru_RU.UTF-8"
  LC_NUMERIC="ru_RU.UTF-8"
  LC_TIME="ru_RU.UTF-8"
  LC_COLLATE="ru_RU.UTF-8"
  LC_MONETARY="ru_RU.UTF-8"
  LC_MESSAGES="ru_RU.UTF-8"
  LC_PAPER="ru_RU.UTF-8"
  LC_NAME="ru_RU.UTF-8"
  LC_ADDRESS="ru_RU.UTF-8"
  LC_TELEPHONE="ru_RU.UTF-8"
  LC_MEASUREMENT="ru_RU.UTF-8"
  LC_IDENTIFICATION="ru_RU.UTF-8"
  LC_ALL=
```

Таким образом, в текущей конфигурации и без принятия дополнительных мер, настройки всех категорий локали совпадают.

Также используются директории (*.mo файлы):
`/etc/share/locale/`
`/etc/share/locale-bundle/`
`/etc/share/locale-langpack/`

Для других Unix-систем, в т.ч. и для других дистрибутивов, размещение файлов может отличаться. Например, для семейства RedHat Linux используется файл:
`/etc/sysconfig/i18n`

6.4.4 Обеспечение интернационализации/локализации в программах

Общая идеология:
– переменные локали установлены для всей системы в целом;
– программа может выбрать одну или несколько локалей в качестве активных;
– программа пользуется только теми предоставляемыми функциями, которые учитывают текущие региональные настройки.

Таким образом, ответственность за результат распределяется:
– программа – сам факт использования локали и выбор ее версии
– система (библиотеки) – техническое исполнение выбранной настройки

Это касается только первого, нижнего уровня локализации. Перевод текстовых сообщений и иных данных выполняется разработчиками программы, программа лишь выбирает нужный файл сообщений.

Поддержка в библиотеке (для языка C)

`/usr/include/locale.h` – заголовочный файл, отвечающий за использование функций локализации. Он содержит объявления:

– структура **`lconv`** – сведения о текущей кодировке и описание визуализируемого формата

– макросы (константы) категорий

– прототипы функций **`localeconv()`** (получение информации о локали) и **`setlocale()`** (установка локали или отдельных категорий)

Функция **`setlocale()`** действует только на текущую программу. Помимо стандартизованных языковых тегов, предусмотрены два особых значения «кодировок»:

– пустое значение – будет использована «локализация по умолчанию», определяемая общесистемными настройками (и в т.ч. переменными окружения)

– значение «**`C`**» – стандартная минимальная локализация для окружения языка C

До ее вызова действует значение «**`C`**», т.е. стандартные для языка C параметры. В исходных текстах программ всегда соблюдаются стандартные для окружения C соглашения о форматах (требование спецификаций POSIX).

Уровни локализации со стороны программы:

1) 8-разрядное представление символов, но для всех операций над ними использует «локалезависимые» средства (например, функции из **`ctype.h`**) вместо обычной «числовой» арифметики и сравнений. Необходимо установить локаль, например в простейшем случае:

```
setlocale( LC_ALL, "")
```

т.е. все настройки локали переопределяются с «**C**» на платформозависимые – установленные для системы в целом.

2) Поддержка и переключение форматов, методов сравнения (сортировки), единиц измерений и т.п. Использование функций **`strcoll()`**, **`strxfrm()`**, **`localtime()`**, **`strftime()`**, а также анализ параметров с помощью **`localeconv()`** для «ручного» форматирования сложных (не имеющих готовой поддержки в форматном вводе-выводе) видов данных.

3) Вынесение всех текстовых сообщений в файлы сообщений **`*.mo`** («каталоги»), размещение их в соответствующих поддиректориях (имена директориев и файлов «каталогов» соответствуют языковым тегам), использование с помощью соответствующих функций.

– **`gettext()`** и др. – выбор сообщения по его идентификатору; в качестве идентификатора используется строка – как правило, это сам текст в версии стандартной латиницы; «базовый» директорий для каталогов задается функцией **`bindtextdomain()`**

– **`catgets()`** – выбор сообщения по идентификатору из заранее открытого «каталога»

– **`catopen()`** и **`catclose()`** – открытие и закрытие «каталога» (файла сообщений). При открытии может быть явно указан абсолютный путь к файлу каталога, либо он ищется в соответствии с текущими настройками (категория локали **`LC_MESSAGES`**).

4) Использование Unicode вместо 8-разрядной кодировки символов (тип **`wchar_t`** вместо **`char`**). Предусмотрен ряд функций, поддерживающих тип **`wchar_t`**:

– **`fputwc()`** – вывод одного wchar-символа, аналог **`fputc()`**

– **`wprintf()`** – семейство функций форматного вывода, аналогичных семейству **`printf()`**

– **`btowc()`**, **`mbtowc()`**, **`mbrtwc()`** и др. – преобразование типа

и т.д.

Пример: использование переменных локали для программ на C и awk

Программа на C:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <locale.h>

int main( int argc, char** argv)
{
  char sz_msgbuf[128];
  time_t cnt_time;
  struct tm timebuf;
```

```
  setlocale( LC_TIME, ""); //switch from "C" locale
 //to default (defined by external variables)
  cnt_time = time( NULL); //current time as counter
  localtime_r( &cnt_time, &timebuf); //unpack
 //to structured representation
  strftime( sz_msgbuf, sizeof(sz_msgbuf),
   "%x %X", &timebuf); //as string
  puts( sz_msgbuf);
  strftime( sz_msgbuf, sizeof(sz_msgbuf),
   %a %b %d %H:%M:%S %Y", &timebuf); //as string
  puts( sz_msgbuf);
  return 0;
 }
```

Скрипт shell для демонстрации:

```
export LC_TIME="ru_RU.UTF-8"
echo "Use \"Rusian Rusian\" datetime locale \
settings: $LC_TIME"
./test_locale

export LC_TIME="en_UK"
echo "Use \"United Kingdom English\" datetime \
locale settings: $LC_TIME"
./test_locale

export LC_TIME="en_GB"
echo "Use \"Great Britain English\" datetime \
locale settings: $LC_TIME"
./test_locale
```

```
export LC_TIME="en_US"
echo "Use \"United States English\" datetime \
locale settings: $LC_TIME"
./test_locale

export LC_TIME="fr_FR"
echo "Use \"France France\" datetime locale \
settings: $LC_TIME"
./test_locale
```

Демонстрация с использованием awk:

```
export LC_TIME="ru_RU.UTF-8"
echo "Use \"Rusian Rusian\" datetime locale \
settings: $LC_TIME"
awk 'BEGIN { print strftime() }'

export LC_TIME="en_UK"
echo "Use \"United Kingdom English\" datetime \
locale settings: $LC_TIME"
awk 'BEGIN { print strftime() }'

export LC_TIME="en_GB"
echo "Use \"Great Britain English\" datetime \
locale settings: $LC_TIME"
awk 'BEGIN { print strftime() }'
```

```
export LC_TIME="en_US"
echo "Use \"United States English\" datetime \
locale settings: $LC_TIME"
awk 'BEGIN { print strftime() }'

export LC_TIME="fr_FR"
echo "Use \"France France\" datetime locale \
settings: $LC_TIME"
awk 'BEGIN { print strftime() }'
```